Trading Strategy Backtester

A comprehensive backtesting framework implementing the four-step validation process outlined in "How I Develop Trading Strategies" by Damian Lillard. This system provides rigorous validation of trading strategies through:

1. In-sample optimization
2. In-sample Monte Carlo permutation test
3. Walk-forward testing
4. Walk-forward Monte Carlo permutation test

Project Structure

Copy

```
backtester/
│
├── strategies/
│   ├── base.py              # Abstract base class for strategies
│   ├── donchian_breakout.py # Donchian Channel Breakout strategy
│   ├── moving_average_crossover.py # Moving Average Crossover strategy
│
├── data/
│   ├── btc_hourly.csv       # Sample data file
│
├── engine/
│   ├── backtester.py        # Main backtesting engine
│   ├── strategy_factory.py  # Factory for creating strategy instances
│   ├── walk_forward.py      # Walk-forward optimization implementation
│
├── utils/
│   ├── metrics.py           # Performance metrics calculation
│   ├── plot.py              # Visualization utilities
│   ├── data_loader.py       # Data loading utilities
│   ├── permutation_test.py  # Permutation test implementation
│   ├── config.py            # Configuration handling
│
├── run_backtest.py          # Main script to run backtests
├── config.yaml              # Configuration file
```

Installation

1. Clone this repository:

bash Copy

```bash
git clone https://github.com/yourusername/trading-strategy-backtester.git
cd trading-strategy-backtester
```

2. Install the required dependencies:

bash Copy

```bash
pip install -r requirements.txt
```

Usage

Basic Usage

To run a backtest with all four validation steps:

bash Copy

```bash
python run_backtest.py --strategy donchian_breakout
```

To run specific validation steps only (e.g., steps 1 and 2):

bash Copy

```bash
python run_backtest.py --strategy donchian_breakout --steps 1,2
```

Configuration

Edit the `config.yaml` file to customize:

- Data settings
- Validation parameters
- Strategy-specific settings
- Output preferences

Creating a New Strategy

1. Create a new strategy file in the `strategies/` directory (e.g., `strategies/my_strategy.py`)
2. Implement your strategy by inheriting from the `Strategy` base class
3. Implement the required methods: `generate_signals()` and `optimize()`
4. Register your strategy in `engine/strategy_factory.py`
5. Add your strategy configuration to `config.yaml`

Example:

python Copy

```python
from strategies.base import Strategy
import numpy as np
import pandas as pd

class MyStrategy(Strategy):
    def __init__(self, param1=10, param2=20):
        super().__init__(name="My Strategy")
        self.parameters = {
            "param1": param1,
            "param2": param2
        }

    def generate_signals(self, data, parameters=None):
        # Implementation of signal generation
        # ...
        return signals

    def optimize(self, data, objective_func, parameter_grid):
        # Implementation of parameter optimization
        # ...
        return best_params, best_value
```

Then in `strategy_factory.py`:

python

Copy

```python
from strategies.my_strategy import MyStrategy

STRATEGY_REGISTRY = {
    # ...
    'my_strategy': MyStrategy,
}
```

Validation Steps

1. In-Sample Optimization

This step finds the best strategy parameters on the training data. The objective is to achieve excellent performance before moving to testing.

2. In-Sample Permutation Test

This test verifies that the strategy's performance is due to actual patterns in the data rather than data mining bias. It uses Monte Carlo simulations with permuted price data to determine if the strategy's performance could be achieved by chance.

3. Walk-Forward Optimization

This test evaluates the strategy's ability to perform in out-of-sample conditions by periodically re-optimizing parameters on a sliding window of data.

4. Walk-Forward Permutation Test

This test verifies that the walk-forward performance is due to the strategy exploiting genuine market patterns rather than chance. It uses Monte Carlo simulations with permuted out-of-sample data.

Output

- Terminal output showing performance metrics for each validation step
- Visual plots of equity curves and performance metrics
- JSON files with detailed results

License

MIT